RMAC: Runtime Configurable Floating Point Multiplier for Approximate Computing

Mohsen Imani, Ricardo Garcia, Saransh Gupta, and Tajana Rosing
CSE Department, UC San Diego, La Jolla, CA 92093, USA
{moimani, rag023, sgupta, tajana}@ucsd.edu

ABSTRACT

Approximate computing is a way to build fast and energy efficient systems, which provides responses of good enough quality tailored for different purposes. In this paper, we propose a novel approximate floating point multiplier which efficiently multiplies two floating numbers and yields a high precision product. RMAC approximates the costly mantissa multiplication to a simple addition between the mantissa of input operands. To tune the level of accuracy, RMAC looks at the first bit of the input mantissas as well as the first N bits of the result of addition to dynamically estimate the maximum multiplication error rate. Then, RMAC decides to either accept the approximate result or re-execute the exact multiplication. Depending on the value of N, the proposed RMAC can be configured to achieve different levels of accuracy. We integrate the proposed RMAC in AMD southern Island GPU, by replacing RMAC with the existing floating point units. We test the efficiency and accuracy of the enhanced GPU on a wide range of applications including multimedia and machine learning applications. Our evaluations show that a GPU enhanced by the proposed RMAC can achieve 5.2× energy-delay product improvement as opposed to GPU using conventional FPUs while ensuring less than 2% quality loss. Comparing our approach with other state-of-the-art approximate multipliers shows that RMAC can achieve 3.1× faster and 1.8× more energy efficient computations while providing the same quality of service.

CCS CONCEPTS

• **Hardware** → **Integrated circuits**; • **Computer systems organization** → *Architectures*; • **Computing methodologies** → Machine learning;

KEYWORDS

Approximate Computing, Floating Point Multiplications, Deep Neural Network, Energy Efficiency

ACM Reference Format:
Mohsen Imani, Ricardo Garcia, Saransh Gupta, and Tajana Rosing. 2018. RMAC: Runtime Configurable Floating Point Multiplier for Approximate Computing. In *ISLPED '18: ISLPED '18: International Symposium on Low Power Electronics and Design, July 23–25, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3218603.3218621

1 INTRODUCTION

The number of smart devices already exceeds the total number of human beings in the world [1]. As Internet of Things (IoT) becomes a reality, humans will be significantly outnumbered by networked devices ready to respond to our every need [2]. In a world in which humans still experience everything through our five senses, the IoT systems in the near future will need to interface with us via embedded devices in many ways similar to how it is done today. The large amount of data generated by IoT challenges our data analysis ability [3, 4]. As a result, embedded devices, such as mobiles, will need to become IoT computing nodes, executing algorithms capable of processing raw sensing streams that traditionally have run on servers. Unfortunately, existing processing cores cannot sustain the computational load required by IoT because the necessary algorithms needed to process IoT data would consume too much battery power, and fails to provide real-time feedback [3]. In addition, it is favorable to keep the data private and locally process it [5].

Many of the algorithms that are run on today's sensor data are at their heart statistical, and thus do not require exact answers [6–11]. Therefore, approximate computing is a promising emerging paradigm for improving energy efficiency of these devices [6, 12, 13]. Approximate computing should be performed on the systems which has less demand on precision, which means the precision could be dropped for energy conserving purposes. Even though approximation is not a completely new idea, as it has been widely used in areas such as lossy compression and numeric computation [14], it is currently gaining a lot of attention in many research areas.

The development of the Internet of Things will bring a new class of applications driven by machine learning that can exploit approximations [15]. Multiplication is one of the most important and common operations that usually happens in floating point representation. However, the FPUs are slow and costly in today's system. Although there were several attempts to design an approximate multiplier, they are not capable of changing the level of accuracy at runtime [16–20]. In addition, the hardware that provides dynamic configurability gives poor accuracy with coarse grain tuning capability [21]. This limits the number of applications that could benefit from approximation.

In this paper, we introduce a runtime configurable floating point multiplier, called RMAC, that can adapt to the accuracy requirement of each application. RMAC replaces the costly floating point multiplication with the addition of the two input mantissa. The reasoning behind replacing the multiplication with addition comes from looking at the core of the multiplication, which is just a shifting and adding process that takes place. In order to adaptively control the accuracy of RMAC, we use the first N bits of the approximate mantissa in order to configure the level of approximate. Based on the estimated error rate, RMAC can accept the approximate result or re-execute the multiplication in exact hardware. This process is done by selecting a value of N, where the value of N dictates the level of accuracy. The efficiency and accuracy of the RMAC was tested on a wide range of applications running on by integrating RMAC on AMD Southern Island GPU. The evaluations show that enhanced GPU can achieve 5.2× energy-delay product improvement as opposed to GPU using conventional FPUs while ensuring less than 2% quality loss. Comparing our approach with state-of-the-art approximate multipliers shows that RMAC can achieve 3.1× faster and 1.8× more energy efficiency computation while providing the same quality of service.

ISLPED '18, July 23–25, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5704-3/18/07…$15.00
https://doi.org/10.1145/3218603.3218621

2 RELATED WORK

Approximate computing in hardware can be achieved using different techniques. The first is voltage over scaling (VOS), which dynamically reduces the supply voltage of hardware components for saving energy, at the cost of accuracy [22–25]. Work in [22] modeled timing error of arithmetic units subject to VOS and employed error detection and control at the system level. Work in [23] investigates the trade-off between energy and quality for image/video processing applications subject to VOS. Work in [24] employs dynamic segmentation with multi-cycle compensation for errors introduced by VOS.

The second technique focuses on redesigning the traditional basic blocks. In the design of multipliers, previous work mostly focused on truncated multiplication [16–18, 26]. The idea is to neglect some partial products on multiplication to speed up adder computation. Work in [18] focused on minimizing the area and increasing the speed of multiplication. This design redesign a multiplier to use less logic gates to approximate multiplication. Work in [16] designed an approximate multiplier which finds the first leading 1 (from the most significant position) in both multiplication input operands, then minimizes the size of required multiplier to ensure minimum quality loss. Although these designs improve multiplication efficiency, they cannot be integrated on general purpose processors as they set the multiplier size for each application offline. In addition, these designs are not adaptive or data dependent as they provide the same level of approximation regardless of input data.

Recently, work in [21] proposed CFPU, a configurable floating point multiplier, which multiplies input operands adaptively depending on the input mantissa. CFPU looks at one the input mantissa and accordingly decides to run the multiplication in exact or approximate mode. However, CFPU has the following limitations: (i) it applies aggressive approximation with possibility of 0% to 50% error rate. (ii) It works in approximate mode only when the the input operand mantissa has N leading one or zero bits (N is a tuning bits). This significantly reduces the amount of the values that can be approximated, since in real world workloads, there are very few input operands can satisfy this condition. In contrast, we propose a runtime configurable approximate multiplier that significantly increases the number of operands which can multiply on approximate mode, while providing the similar accuracy as prior work. Our design ensures that the maximum level of multiplication error rate will never surpass 11.1%.

3 PROPOSED DESIGN

3.1 RMAC Overview

Using the IEEE 754 32 bit floating point notation that is represented as a binary number string of 32 bits $(A_{32},\ldots,A_1)$ with three different parts: a sign bit, an exponent part, and a fractional value. The first bit in the floating point notation (A_{32}) represents the sign bit. The next 8 bits represent the exponent of the binary numbers $(A_{31},\ldots,A_{24})$, ranging from -126 to 127. The following 23 bits $(A_{23},\ldots,A_1)$) represent the fractional part, also known as the mantissa, which has a value between 1 and 2. Figure 1 (a) shows floating point multiplication between A and B input operands. The process starts by `XOR`ing the sign bits of A and B, adding the exponent of A and B and finally multiplying the two mantissas. To improve the multiplication efficiency, we need to accelerate the mantissa multiplication process as this operation is the most costly operation.

In this paper, we propose a runtime configurable floating point multiplier, called RMAC, that supports multiplication in both exact and approximate modes. In exact mode, the input values are multiplied using the IEEE-754 32 bit standard method. Figure 1 (b) the approximate model of our proposed design, where mantissa

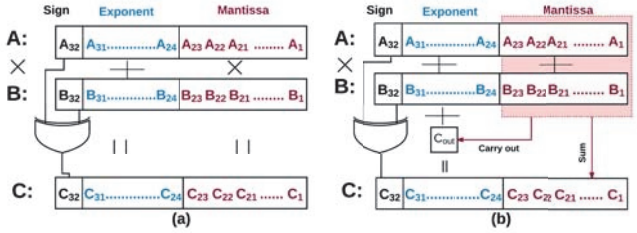


Figure 1: The multiplication between A and B floating point values in (a) IEEE-745 32 bit standard and (b) the proposed RMAC

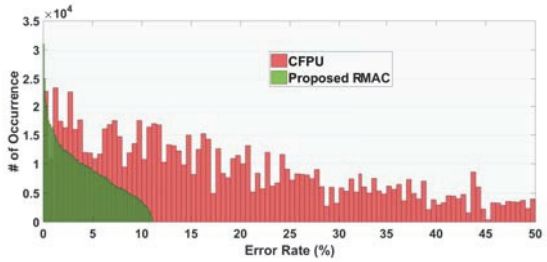


Figure 2: Histogram of the error distribution of the CFPU and proposed RMAC.

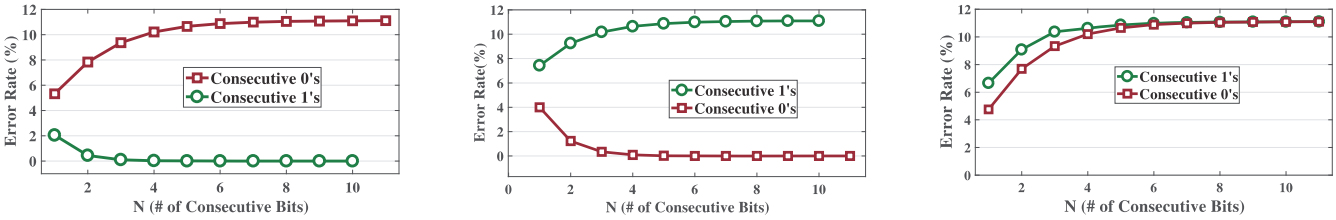
multiplication is replaced with the addition of the mantissas. In case of an overflow in addition, the extra carry bit is added to the exponent. This is a suitable approximation to fixed-point multiplication between the mantissas since it can be done by addition and shifting of the operation between the partial products. In floating point representation, the shift is already applied by normalizing the mantissas values to a value between 1-2, therefore, this addition can be a good approximation of the mantissa multiplication.

Figure 2 compares the error distribution of the proposed RMAC with CFPU [21], the state-of-the-art approximate FPU, by multiplying 1 million random floating point numbers together. For both designs, the figure shows the histogram of the multiplication error. The result shows that RMAC has the maximum quality loss of 11.1% whereas the CFPU error can be as large as 50%, which is about 5× higher than the error rate that RMAC can achieve.

Although, without having a proper tuning process the RMAC's error rate will vary by its input operands. For example, if A=5 and B=10 using floating point multiplication the exact result is 50. Using the RMAC the approximation is 48, this results in a 4% error. However, if A=12 and B=12 in exact mode the product is 144, in approximation mode the result is 128, this results in an 11.1% error. Therefore, a tuning process that limits the error rate of the RMAC will allow the user to dictate how much error is acceptable. Thus, approximation mode will only run, if the product of the two input operands is below a certain threshold.

3.2 Accuracy Tuning

In the real world, applications have different sensitivity to approximation. Even within a single application, different parts of codes might have different sensitivity to approximation. In order to generalize RMAC on general purpose processors, such as CPU and GPU, RMAC requires being adaptive and data dependent. In fact, RMAC should be able to control the level of accuracy depending on the running application. Sine the execution time of mantissa addition and error estimation is significantly lower than the execution time of exact mantissa multiplication the error rate for approximation



(a) Input mantissas (1, 1) (b) Input mantissas bits (0, 0) (c) Input mantissas (0, 1)

Figure 3: The maximum error of the RMAC depending on the first N bits sequence of the approximated mantissa.

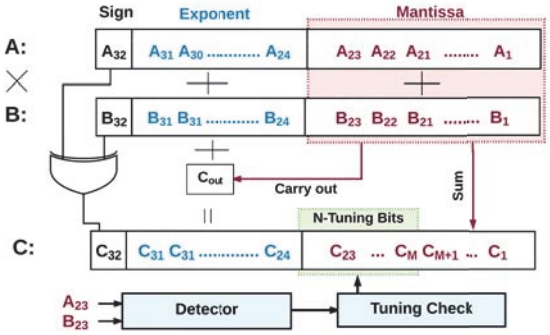


Figure 4: Tuning process of the proposed RMAC depending on the input operand mantissa bits.

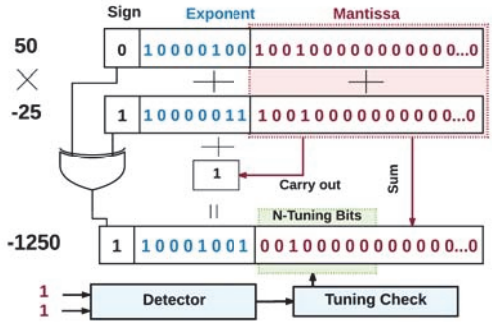


Figure 5: This example shows the tuning process for the proposed RMAC.

can be checked for the desired error rate. However, if the desired error rate is not meet the multiplication can be redone in exact mode with minimal speed and energy consumption drawbacks. In order to find a tuning process, every possible value of A was multiplied with every possible value of B for the IEEE 754 8, 16, and 32 bit floating point notation. Through this experiment, the result shows that depending on the first bit of the input mantissas, RMAC follows different approximation trends. Figure 3 shows the result of the 32-bit evaluation performed to determine the sensitivity of the RMAC approximation to input operands. The result also shows that as the number of consecutive 1's and 0's increases the error rate decreases.

Figure 4 shows how the tuning process works. RMAC starts by detecting the values of the leading mantissa bit of both input operands. Based on the values of A_{23} and B_{23}, the RMAC will perform one of the following techniques to tune the error rate:

3.2.1 Case 1. When both A_{23} and B_{23} are 1, the error rate of RMAC varies mostly with respect to the number of consecutive 0's in mantissa C (shown in Figure 3a). RMAC tunes the level of accuracy by looking at the number of consecutive 0's in the answer's mantissa. Depending on the desired accuracy, the number of consecutive 0's changes. In this specific case, the maximum error is 11.1% when the answer's mantissa is a full sequence of 0's and a minimum error of 5.53% when the mantissa has a leading 0 followed by a 1.

3.2.2 Case 2. When both A_{23} and B_{23} are 0, the accuracy is determined by the number of consecutive 1's in the answer's mantissa similar to the first case. Figure 3b shows the the error rate of RMAC when mantissa C contains N consecutive 1's. In this case the maximum error of 11.10% when the answer's mantissa is a full sequence of 1's and a minimum error of 7.43% when the mantissa has a leading 1 followed by a 0.

3.2.3 Case 3. When both A_{23} and B_{23} are different values, the value of C_{23} is considered. If C_{23} is a 0 or 1, the number of consecutive 0's or 1's in the answers mantissa respectively dictate the accuracy. Figure 3c shows the error rate of RMAC when C contains N continuous 1's or 0's. For this case, there are two different maximum and minimum errors because C_{23} can either be 1 or 0. When C_{23} is 1, the maximum error is 11.10% when the mantissa C contains all 1's and a minimum of 6.66% when it has a leading 1 followed by a 0. When C_{23} is 0 the maximum error is 11.11% when mantissa C contains all 0's and a minimum of 4.76% when the mantissa has a leading 0 followed by a 1.

The overall trend across all cases illustrates that as the number of consecutive 1's or 0's decreases, the accuracy of the approximation increases. Figure 5 shows an example of how the tuning process works by multiplying two floating numbers (50 and -25) together. Since both A_{23} and B_{23} are equal to 1, the design looks for the number of consecutive 0's in mantissa C represented by C_{23} to C_1. This approximation has an error rate of 7.84% because mantissa C has two consecutive 0's which corresponds with the results of Figure 3a. Most importantly, this example illustrates how the RMAC can be tuned for a certain degree of accuracy by selecting different N values. For example, if the application requires more accuracy (by having N=1), then this case would run in exact mode. On the other hand, if the application allows for a broader range of error (by having N=5), then this case would run in approximation mode.

Figure 6 shows the error distribution of the RMAC error when it uses different tuning bits, N. As figure shows, while using no tuning bits the maximum RMAC error rate will be 11.1%. Using $N = 3$ tuning bit, this maximum error rate decreases to 10.7%. Similarly, further decreasing the N value to 2 and 1 reduces the maximum multiplication error rate to 9.3% and 6.8%.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

We integrated the proposed RMAC in a GPU southern Island architecture, Radeon HD 7970 device. We modified Multi2sim, a cycle accurate CPU-GPU simulator [27] to model the RMAC functionality

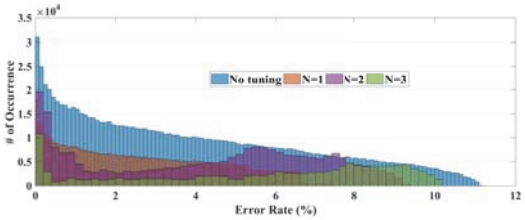


Figure 6: RMAC error distribution on different tuning bits.

Table 1: Comparison of the proposed RMAC with state of the art approximate multipliers.

	Kulkarni[18]	ESSM8 [17]	DRUM6 [16]	CFPU3 [21]	RMAC
Tunable	No	No	No	Yes	Yes
Max Error	22.2%	11.1%	6.3%	6.3%	6.3%
Energy (pJ)	2.9	0.58	0.55	0.27	0.15
Execution (ns)	3.5	2.1	1.9	1.6	0.52

Table 2: CFPU and RMAC Hit rate and EDP improvement while multiplying 1 million randomly generated numbers.

Error Rate		1%	2%	4%	6%	10%	12%
CFPU[21]	*Hit Rate*	3.4%	7.5%	14.7%	19.4%	31.1%	37.2%
	EDP Improv.	1.03×	1.07×	1.16×	1.22×	1.41×	1.54×
RMAC	*Hit Rate*	25.1%	42.7%	71.2%	93.7%	99.9%	100%
	EDP Improv.	1.29×	1.63×	2.84×	6.80×	11.07×	11.31×

on three main floating point units in GPU architecture: MUL, MAC and MAD. The FPUs are balanced for 6-stage using FloPoCo [28] and are synthesized by *Synopsys Design Compiler* in 45-nm ASIC flow and optimized for power consumption using *Synopsys Prime Time*. The energy consumption and execution time of the proposed RMAC are measured using circuit level simulation, using HSPICE simulator in 45-nm technology. For the application level, we test the efficient of the proposed RMAC by running wide range of applications on GPU as listed below:

OpenCL: We selected four general Open*CL* applications including *Sobel, Roberts, HwtHaar1D* and *BinomialOption* from AMD APP SDK v2.5 [29]. For image processing applications, we use Caltech 101 [30] as our dataset, while for other applications we generate the dataset using random generator. For these applications we designed average relative error as a quality metric.

Rodinia: We also examine our design on four Rodinia 3.1 benchmark suite [31], including *K-means*, *Back Propagation, Lud* and *K-Nearest Neighbor(KNN)*,*K-means* and *K-nearest neighbor* are used in data mining applications and involve dense linear algebra computation, while *Back Propagation* is used for training weights in neural networks. For Rodinia applications (*K-means* and KNN), the quality loss defines as the rate of miss classification or miss clustered points as compared to the same application running on exact hardware.

Neural Network: We evaluate RMAC efficiency on the three neural network (NN) applications. NNs are realized using OpenCL, an industry-standard programming model for heterogeneous computing. We used stochastic gradient descent with momentum of 0.1, learning rate of 0.001 and batch size of 10 for the NN training. The activation functions are set to "Rectified Linear Unit" clamped at 6. A "Softmax" function is applied to the output layer. For neural network, the quality loss defines as the difference of classification accuracy running each application on exact and approximate hardware. The tested applications are listed as follows:

Hand Writing Recognition (MNIST): MNIST is a popular machine learning data set including images of handwritten digits [32]. The objective is to classify an input picture to one of the ten digits {0 . . . 9}.

Human Activity Recognition (HAR): HAR recognizes human activity based on 3-axial linear acceleration and 3-axial angular velocity that have been captured at a constant rate of 50Hz [33].

Voice Recognition (ISOLET): ISOLET [34] consists of speech collected from 150 speakers. The goal of this task is to classify the vocal signal to one of the 26 English letters.

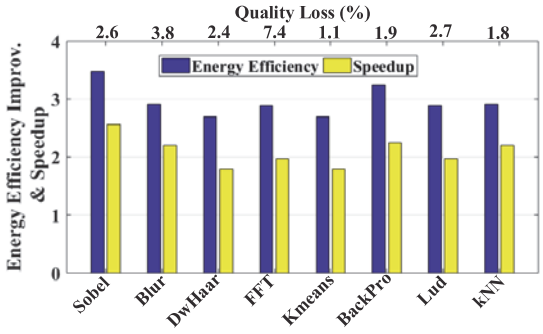


Figure 7: Energy efficiency improvement, speedup and quality loss of different applications running on enhanced GPU (RMAC with no tuning).

4.2 Comparison

We compare the energy efficiency and execution time of the proposed RMAC with the state-of-the-art approximate multipliers including: ESSM [17], Kulkarni[18], DRUM [16], and CFPU [21]. Looking at different designs, we can see that CFPU and proposed RMAC are the only approximate multipliers with runtime tuning capability. Other multipliers can configure the level of approximation at offline, thus they are not data dependent. Table 1 lists the energy efficiency and execution time of different multipliers in their best configurations when all designs ensure the similar maximum error rate. Since CFPU and the proposed RMAC are data dependent, we consider their average energy and execution time on 1 million randomly generated data. Our evaluation shows that when the RMAC provides the same level of error rate (using $N = 1$), it can provide at least 2.1× higher energy efficiency and 1.7× faster execution as compared to prior designs. This efficiency mostly comes from the higher percentage of values that the proposed RMAC can approximately process, compared to CFPU.

As both CFPU and proposed RMAC are tunable, we compare their advantages in more details. Table 2 lists the hit rate and the EDP improvements of the CFPU and proposed RMAC multiplication running 1 million randomly generated values. The results are reported when both designs ensure less than a certain amount of error rate over all tested data. Hit rate is defined as the average time each multiplier is used in approximate mode to the total number of multiplications. The EDP improvements are relative to the conventional floating point multipliers. Our evaluation shows that RMAC can achieve significantly higher hit rate and efficiency as compared to CFPU while providing the same level of computation quality. For example, ensuring less than 6% error rate, the CFPU and RMAC provide 19.5% and 93.7% hit rates and 1.22× and 6.8× EDP improvement as compared to the conventional FPU. The difference of CFPU and RMAC efficiency grows by increasing the level of acceptable error rate. This is because the proposed RMAC with no tuning always ensures less than 11.1% error rate over any data. Therefore,

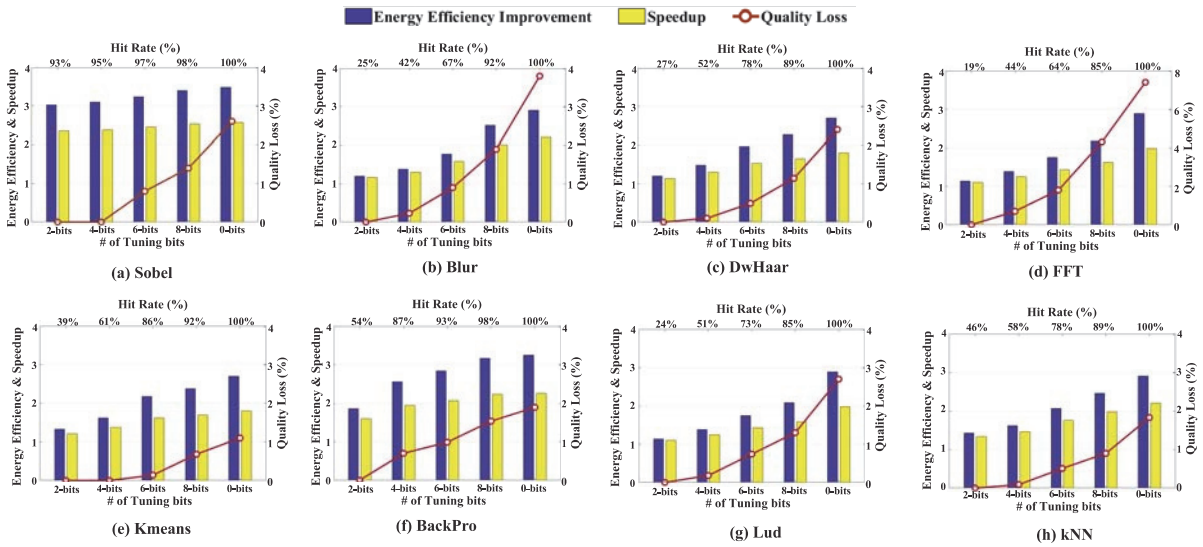


Figure 8: Energy consumption and execution time of RMAC in different tuning bits. x-axis is N tuning bits and the left y-axis is normalize energy and execution, the right axis is accuracy

Table 3: Configuration and energy-delay product improvement of the applications in different computation quality loss.

Quality Loss		Sobel	Blur	DwHaar	FFT	K-means	BackPro	Lud	kNN	Average
0%	*N Tuning*	4-bits	2-bits	2-bits	-bits 2-bits	4-bits	2-bits	2-bits	2-bits	**2.47×**
	EDP Improv.	7.37×	1.38×	1.36×	1.25×	2.22×	2.99×	1.25×	1.91×	
1%	*N Tuning*	6-bits	6-bits	6-bits	4-bits	8-bits	4-bits	6-bits	8-bits	**3.98×**
	EDP Improv.	7.94×	2.81×	3.01×	1.71×	4.02×	4.97×	2.50×	4.87×	
2%	*N Tuning*	8-bits	8-bits	8-bits	6-bits	0-bits	0-bits	8-bits	0-bits	**5.23×**
	EDP Improv.	8.59×	5.08×	3.76×	2.50×	4.86×	7.31×	3.31×	6.43×	

when an application can accept more than 11.1% maximum error rate, the RMAC hit rate increases to 100%, meaning that RMAC can be used with no tuning scheme. However, the CFPU error rate can increase up to 50% during the approximation. In addition,

4.3 Efficiency in Application Level

Figure 7 shows the energy consumption and execution time of different applications on enhanced GPU normalized to GPU using conventional FPUs. Considering RMAC with no tuning capability, our evaluation shows that GPU can provide acceptable quality of service over most of the workloads. Our results show that the *Rodinia* applications have higher robustness to the RMAC approximation, as several *Rodinia* benchmarks are stochastic and approximate in heart. Instead, the applications such as digital filters show higher quality loss using approximate RMAC units. In average, our evaluations indicates that GPU using RMAC can achieve 3.0× and 2.9× higher energy efficiency and 2.3× and 1.9× speedup on general *OpenCL* and *Rodinia* applications while ensuring less than 7.4% quality loss.

4.4 Efficiency-Accuracy Trade-off

In order to keep the GPU generality, our enhanced GPU should be able to accelerate general applications. Therefore, RMAC needs to be a able to tune the level of accuracy based on the applications sensitivity and users requirement. Figure 8 shows the impact of the tuning bits on the quality of different applications. In this figure, the x-axis shows the number of tuning bits, while the y-axis shows the normalized energy consumption and execution time of the RMAC. As our results show, applications have different sensitivity to approximation and they provide the same quality of service on different RMAC configurations. For instance, to ensure less than 1% quality loss, application such as *Sobel* requires four tuning bits, while the machine learning application such as *k-means* can satisfy 1% quality loss using no tuning scheme.

Table 3 lists the configuration and energy-delay product improvement that each application can achieve while losing different level of quality. To provide less than 1% quality loss, Rodinia benchmarks can work on more approximation level (lower tuning bits) as compared to general deterministic OpenCL benchmarks. Accepting 1% quality loss, our evaluation shows that enhanced GPU can achieve 2.1× energy efficiency improvement and 1.7× speedup (3.6× EDP improvement) as compare to GPU using conventional FPUs. Ensuring less than 2% quality loss, this efficiency increases to 2.6× in energy and 2.0× in execution time (5.2× EDP improvement).

4.5 RMAC for Neural Network Acceleration

Floating point multiplication is one of the essential operations in neural network (NN) applications. The efficiency and high error resiliency of the proposed RMAC make it a suitable candidate for NN acceleration. In this part, we explore the impact of the RMAC on three popular NN applications: Handwriting digits (MNIST), Speech Recognition (ISOLET), and Human Activity Recognition (HAR). For each application, we used well-known network architecture suggested by the Keras library. Table 4 lists the configuration of the network and their baseline accuracy. We run these applications on the conventional and enhanced AMD southern Island GPU using Multi2sim simulator. As Table 4 shows, the enhanced GPU without tuning results in about 2% quality loss over different applications, while it provides in average 2.6× speedup and 4.1× energy efficiency improvement.

The tuning capability of the RMAC gives us more opportunity to improve the NN efficiency. Looking at NN functionality, all NN layers do not have the same sensitivity to approximation. For instance, in NN the middle hidden layers show significantly higher

Table 4: Baseline network topologies, the baseline accuracy and the quality loss running on enhanced GPU (no tuning).

Application	Network Topology (l^0,l^1,l^2,l^3)	Baseline Accuracy	Quality Loss (RMAC N=0)
Handwritten Digit (MNIST)	784, 500, 500, 10	97.6%	1.8%
Activity Recognition (HAAR)	561, 500, 500, 12	96.6%	2.4%
Voice Recognition(ISOLET)	617, 500, 500, 26	95.5%	2.1%

Table 5: The configuration, EDP improvement of enhanced GPU using uniform and selective approximation.

		0%		0.5%		1%	
		Uniform Approx	*Selective Approx*	*Uniform Approx*	*Selective Approx*	*Uniform Approx*	*Selective Approx*
MNIST	**config (N)**	2	2, 4, 4, 2	2	4, 8, 8, 4	6	8, 0, 0, 8
	EDP	1.38×	2.42×	1.38×	4.73×	3.27×	7.23×
ISOLET	**config (N)**	2	2, 2, 4, 2	4	4, 4, 8, 4	4	6, 6, 0, 6
	EDP	1.43×	1.96×	2.67×	3.80×	2.67×	4.82×
HAAR	**config (N)**	2	2, 6, 6, 2	4	6, 8, 8, 6	6	8, 0, 0, 8
	EDP	1.31×	4.54×	2.39×	6.93×	3.22×	7.11×

robustness to approximation as compared to the first and last layers. Therefore, to improve the NN efficiency, our design applies selective approximation to NN layers based on their sensitivity. We perform an experiment to find the sensitivity of different layers to RMAC approximation. To do so, we sequentially put neural network layers on approximate mode (RMAC with no tuning) and measure the impact of each layer approximation on the network classification accuracy. Table 5 shows the configuration of the RMAC in each layer of neural network using uniform and selective approximation. Our evaluation shows that both uniform and selective approximation can perfectly control the classification accuracy of the NN applications by changing the N tuning. However, the selective approximation can achieve significantly higher EDP improvement. For instance, accepting 1% quality loss, enhanced GPU can provide 6.3× EDP improvement as compared to GPU using conventional FPUs. This improvement is 2.1× higher than EDP improvement that uniform approximation can provide.

4.6 Overhead

The conventional 32-bit floating point multiplier takes 7690 μm^2 area. In order to enable RMAC functionality, the conventional multiplier needs to use extra 23-bit fixed-point adder and a tuning circuit. Our evaluation using *Synopsys Design Compiler* shows that the adder and the tuning logic will take 101.5 μm^2 and 28.3μm^2 area respectively. Thus, the RMAC will have a 1.68% larger area as compared to the conventional floating point units. This area overhead is negligible considering the flexibility and efficiency that the RMAC can provide. In worst case when the RMAC multiplications needs to re-execute on the exact mode, RMAC will add 2.1% and 1.7% energy and performance overhead to conventional floating point multiplier. Our evaluations show that RMAC can compensate this overhead by running at least 3.6% of the multiplication on approximate mode. As we showed in section 4.4, the hit rate of RMAC approximation is much higher than these values, since RMAC ensures less than 11.1% error rate over any input data.

5 CONCLUSION

In this paper, we propose a novel approximate multiplier which efficiently multiplies floating point values with significantly less quality loss. RMAC models the costly mantissa multiplication approximately with a simple addition between the mantissa of input operands. To tune the error rate, RMAC looks at the first bit of the input mantissas as well as the first N bits of the added mantissa to dynamically estimate the maximum multiplication error rate. Then, RMAC decides to either accept the mantissa addition or re-execute the exact multiplication. Depending on the N value, the proposed RMAC can be configured to achieve a different level of accuracy. Our evaluations on wide range of applications show that GPU enhanced by the proposed RMAC can achieve 5.2× energy-delay product improvement as compared to GPU using conventional FPUs while ensuring less than 2% quality loss.

ACKNOWLEDGMENT

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] A. Fehske *et al.*, "The global footprint of mobile communications: The ecological and economic perspective," *IEEE Communications Magazine*, vol. 49, no. 8, 2011.
[2] F. Xia *et al.*, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, p. 1101, 2012.
[3] J. Gubbi *et al.*, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[4] B. Yao *et al.*, "Multifractal analysis of image profiles for the characterization and detection of defects in additive manufacturing," *Journal of Manufacturing Science and Engineering*, vol. 140, no. 3, p. 031014, 2018.
[5] M. S. Riazi *et al.*, "Camsure: Secure content-addressable memory for approximate search," *ACM TECS*, vol. 16, no. 5s, p. 136, 2017.
[6] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE ETS*, pp. 1–6, IEEE, 2013.
[7] C. Liu *et al.*, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *IEEE/ACM DATE*, p. 95, IEEE, 2014.
[8] M. Imani *et al.*, "Resistive configurable associative memory for approximate computing," in *DATE*, pp. 1327–1332, IEEE, 2016.
[9] M. Imani *et al.*, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *IEEE/ACM DATE*, pp. 373–378, IEEE, 2017.
[10] X. Jiao *et al.*, "Energy-efficient neural networks using approximate computation reuse," in *DATE*, pp. 1223–1228, IEEE, 2018.
[11] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *IEEE TETC*, 2016.
[12] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *IEEE/ACM ICCAD*, pp. 25–32, IEEE, 2017.
[13] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *DAC*, p. 6, ACM, 2017.
[14] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.
[15] F. Imani and other, "Factal pattern recognition of image profiles for manufacturing process monitoring and control," in *ASME MSEC*, p. 1, 2018.
[16] S. Hashemi *et al.*, "tldrum: A dynamic range unbiased multiplier for approximate applications," in *ICCAD*, pp. 418–425, IEEE Press, 2015.
[17] S. Narayanamoorthy *et al.*, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *TVLSI*, vol. 23, no. 6, pp. 1180–1184, 2015.
[18] P. Kulkarni *et al.*, "Trading accuracy for power with an underdesigned multiplier architecture," in *IVLSI*, pp. 346–351, IEEE, 2011.
[19] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning," in *IEEE/ACM ISLPED*, pp. 162–167, 2016.
[20] M. Imani *et al.*, "Canna: neural network acceleration using configurable approximation on gpgpu," in *IEEE ASPLOS*, IEEE, 2018.
[21] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *DAC*, pp. 1–6, IEEE, 2017.
[22] D. Jeon *et al.*, "Design methodology for voltage-overscaled ultra-low-power systems," *TCAS II*, vol. 59, no. 12, pp. 952–956, 2012.
[23] K. He *et al.*, "Circuit-level timing-error acceptance for design of energy-efficient dct/idct-based systems," *TCSVT*, vol. 23, no. 6, pp. 961–974, 2013.
[24] D. Mohapatra *et al.*, "Design of voltage-scalable meta-functions for approximate computing," in *DATE*, pp. 1–6, IEEE, 2011.
[25] M. Imani *et al.*, "Remam: low energy resistive multi-stage associative memory for energy efficient computing," in *IEEE ISQED*, pp. 101–106, IEEE, 2016.
[26] K. Bhardwaj *et al.*, "Power-and area-efficient approximate wallace tree multiplier for error-resilient systems," in *ISQED*, pp. 263–269, IEEE, 2014.
[27] R. Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *PACT*, pp. 335–344, ACM, 2012.
[28] "Aflopoco [online]. available:http://flopoco.gforge.inria.fr/,"
[29] "Amd app sdk v2.5 [online]. available: http://www.amd.com/stream,"
[30] "Caltech 101 [online]. http://www.vision.caltech.edu/image_datasets/caltech101/,"
[31] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, Ieee, 2009.
[32] Y. LeCun *et al.*, "The mnist database of handwritten digits," 1998.
[33] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones.
[34] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.